ICE2607 Lab 1: Image Feature Extraction

SJTU-SEIEE   cny123222

November 12, 2024

1 Experiment Overview (实验概览)

Lab1 primarily focuses on image feature extraction. Images are processed using OpenCV, and three histograms are generated to show the features of the image.

- Color Histogram (颜色直方图): The color histogram illustrates the relative proportion of the three color components (RGB) in a color image, providing insight into the color distribution and the dominant tone of the image.
- Gray Histogram (灰度直方图): The gray histogram displays the distribution of grayscale values in a grayscale image, offering information about the image's brightness.
- Gradient Histogram (梯度直方图): The gradient histogram shows the distribution of gradient intensities in a grayscale image, indicating the texture sparsity within the image.

2 Solution Approach (解决思路)

Since the experiment mainly revolves matrix operations, **a fully-vectorized method** based on Numpy is designed to extract the required features. This method avoids conventional iterative loops, speeds up the computational process and enhances readability.

2.1 Color Histogram

To compute the color histogram, it is essential to determine the relative proportions of the three color components (RGB). The following pseudocode shows the core of calculating the color distribution:

```
def calc_color(img_path):
    img = cv2.imread(img_path, cv2.IMREAD_COLOR) # img.shape = (H, W, 3)
    color = img.sum(axis=(0, 1)) # color.shape = (3, )
    return color / color.sum()
```

Given the path to the input image, the function first reads the image using OpenCV in color mode, converting it into a Numpy array with a shape of (H, W, 3). Subsequently, the array is summed over the first two dimensions, resulting in a vector (`color`) containing the energy intensities of the three color components. Finally, this vector is normalized to derive the relative proportion of the components, i.e. the RGB distribution.

2.2 Gray Histogram

In computing the gray histogram, it is imperative to determine the frequency of grayscale values ranging from 0 to 255 within the grayscale image. The following pseudocode shows the core of computing the grayscale values distribution:

```
def calc_gray(img_path):
    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE) # img.shape = (H, W)
    img = img.flatten() # img.shape = (H*W, )
    gray = np.bincount(img, minlength=256) # gray.shape = (256, )
    return gray / gray.sum()
```

Upon receiving the path to the input image, the function initially reads the image in grayscale mode, converting it into a Numpy array with a shape of (H, W). Then, the 2D array is flattened and the occurrences of each integer value from 0 to 255 are counted, resulting in a vector (`gray`) denoting the frequency of each grayscale value. Note that np.bincount only accepts one-dimensional arrays. Finally, the vector is normalized to ascertain the relative proportion of the grayscale values.

2.3 Gradient Histogram

When generating gradient histograms, it is crucial to calculate the gradient intensity of the grayscale image. With these intensity values, we can establish the distribution of gradient intensities by using an approach similar to the one we employed to construct gray histograms.

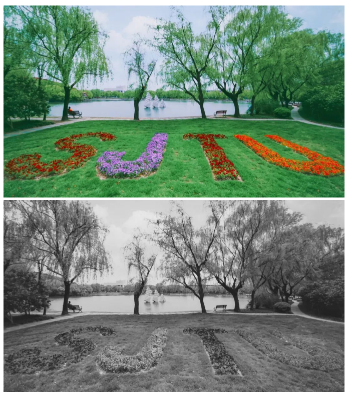
```
def calc_gradient(img_path):
    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE).astype(np.int64) # img.shape = (H, W)
    grad_x = img[1:-1, 2:] - img[1:-1, :-2] # grad_x.shape = (H-2, W-2)
    grad_y = img[2:, 1:-1] - img[:-2, 1:-1] # grad_y.shape = (H-2, W-2)
    grad = np.sqrt(np.pow(grad_x, 2) + np.pow(grad_y, 2)) # grad.shape = (H-2, W-2)
    hist, _ = np.histogram(grad, bins=np.arange(0, 362, 1)) # hist.shape = (361, )
    return hist / hist.sum()
```

Initially, the function reads the image in grayscale mode and converts the array's datatype to int64 (further elaboration will be provided in **Section 5**). Subsequent operations involve calculating gradients along both the x and y axes through Numpy array slicing. It is noteworthy that the outermost pixels are excluded, altering the matrix shape to (H-2, W-2). The gradient intensities are then determined following the definition.

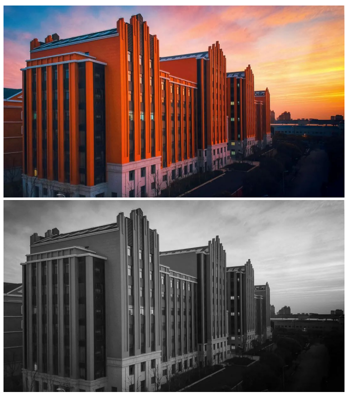
Subsequently, the gradient intensities are tallied within intervals ranging from [0, 1) to [360, 361). Note that the parameter `bins` includes the rightmost edge. Lastly, the vector is normalized to ascertain the distribution of gradient intensities.

3 Experimental Results (实验结果)

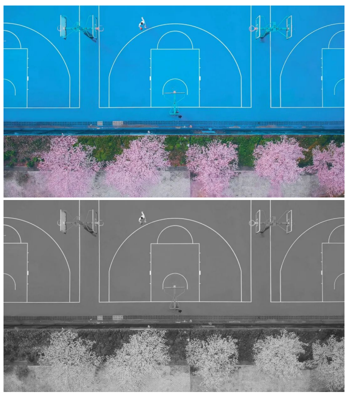
We conducted the experiment on the following three images of SJTU shown in Figure 1. To facilitate a more detailed analysis of our experiment outcomes, we also present the grayscale renditions of the three test images.



(a)



(b)



(c)

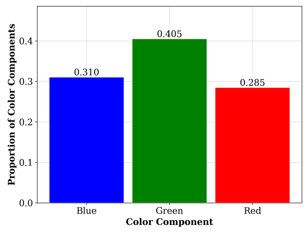
Figure 1　Test Images (Color & Grayscale)

3.1 Experimental Environment

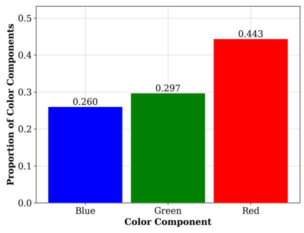
The experiments were conducted on macOS Sonoma 14.6.1 with Python 3.13.0. Key libraries included OpenCV-Python 4.10.0.84, Numpy 2.1.3 and Matplotlib 3.9.2.

3.2 Histogram Results

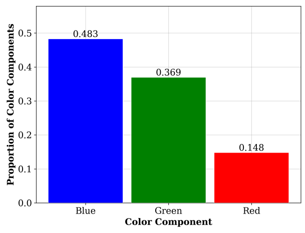
Figure 2, Figure 3, and Figure 4 exhibit the color histograms, grayscale histograms, and gradient histograms of the three test images generated by our algorithms, as per the requirements of the paper.



(a)

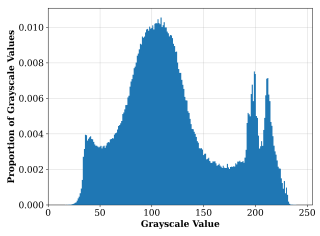


(b)

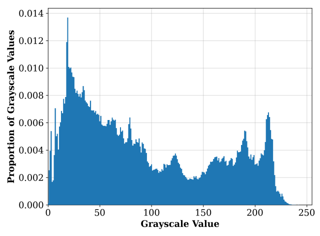


(c)

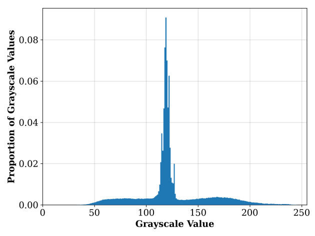
Figure 2　Color Histograms of the Test Images



(a)

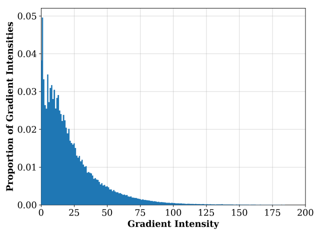


(b)

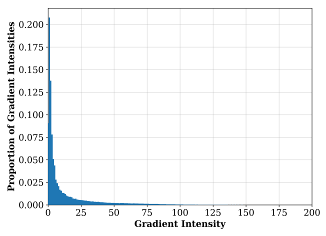


(c)

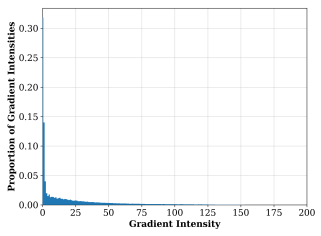
Figure 3 Gray Histograms of the Test Images



(a)



(b)



(c)

Figure 4 Gradient Histograms of the Test Images

4 Analysis and Discussion (分析与思考)

4.1 Analysis of Results

The **color histograms** presented in Figure 2 illustrate the color distribution of the three images and reveal that the predominant colors are respectively green, red, and blue, which is consistent with our intuition. This observation underscores the capacity of color histograms to reflect the overall tonal distribution of an image.

The **gray histograms** depicted in Figure 3 illustrate the distribution of grayscale values across the three images. Given that lighter regions correspond to larger grayscale values, we can infer an image's brightness distribution from its gray histogram. For instance,

- Figure 3(a) shows a prominent peak at a lower grayscale value and a smaller peak at a higher grayscale value. The first peak is likely to correspond to the extensive dark areas comprising grass and trees in Figure 1(a), while the second peak may correspond to the smaller bright regions of lake and sky.
- In Figure 3(c), the grayscale values are highly centralized with minimal variance, contrasting with the dispersed and broad ranges seen in Figure 3(a) and (b). This could be attributed to differences in brightness distributions. In Figure 1(c), brightness is relatively uniform, while in Figure 1(a) and (b), distinct light and dark regions are clearly seen.

The **gradient histograms** showcased in Figure 4 elucidate the distribution of gradient intensities of the three pictures. Notably, the average gradient intensity in Figure 4(a) is evidently higher than that in Figure 4(b) and (c). This observation aligns with our expectations, as Figure 1(a) exhibits lots of texture around the grass and trees, leading to larger gradient variations, while Figure 1(b) and (c) display relatively smoother regions with less textural complexity.

4.2 Further Discussion

4.2.1 Top of Discussion on PowerPoint (PPT 思考题)

- In OpenCV, the default color channel order for images is BGR, whereas in many other libraries like Matplotlib, the color channel order is RGB. To ensure the correct display of the image's color, `cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)` is used to **convert the image from OpenCV's default BGR format to the RGB format expected by Matplotlib**.

- When using Matplotlib's `imshow` function to display a grayscale image, it is crucial to **ensure the colormap is set to 'gray'**. Otherwise, Matplotlib may attempt to use a color colormap to display the grayscale image, leading to misinterpretation of grayscale values as colors.

```
img_gray = cv2.imread('path/to/your/image.jpg', cv2.IMREAD_GRAYSCALE)
plt.imshow(img_gray, cmap='gray') # Set colormap to 'gray'
plt.show()
```

4.2.2 Edge Detection based on Gradient Intensity (基于梯度强度的边缘检测方法)

Edges, characterized by significant pixel value transitions, result in larger gradient intensities. By outputting the gradient intensity image derived in previous experiments, areas with notable intensity fluctuations, particularly edges, appear brighter, offering a creative method for edge detection.

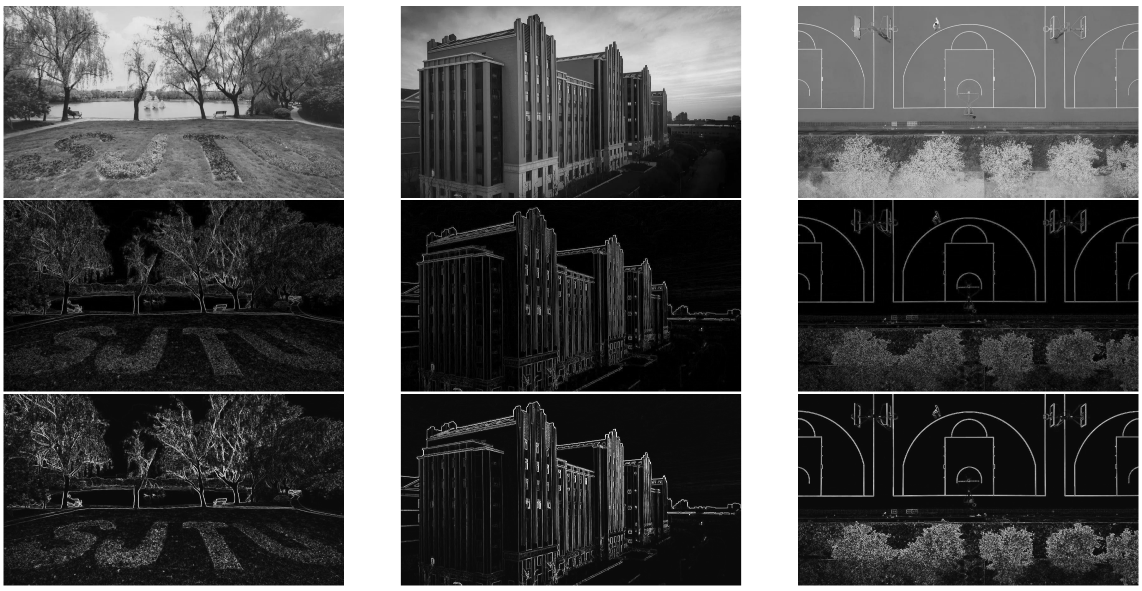


Figure 5 Edge Detection Results (Original Images, Simple Detection, Augmented Detection)

The effect can be further enhanced using a **Sigmoid** function to process the gradient intensities, and the parameters cam be further optimized.

```python
a, b = 80, 25 # parameters can be optimized
grad_img = gradient * (255 / gradient.max()) # Normalization
grad_img = 255 / (1 + np.exp(-(grad_img - a) / b))
```

4.2.3 Highlights of Codes (代码创新点)

- **Object-Oriented Approach:** Utilizing an object-oriented design to enhance code readability and maintainability.
- **Command-Line Interface:** Implementation of a command-line interface allowing for the input of various parameters, enhancing versatility, usability, and cross-system compatibility.
- **Integration of tqdm Progress Bar:** Incorporating a tqdm progress bar to display real-time progress updates and indicate successful saves, improving user experience and task monitoring.
- **Fully-Vectorized Method:** Implementation of vectorization techniques to avoid loops, thereby boosting runtime efficiency and code readability.

5 Reflection and Conclusion (实验感想)

5.1 Learning from Experiments

Through the experiments, we acquired fundamental skills in image feature extraction, basic usage of OpenCV, NumPy, and Matplotlib, as well as LaTeX proficiency, enriching our skill set and fostering creativity.

5.2 Challenges Encountered

- While computing gradient intensity, we encountered overflow issues. Since images are stored in **uint8** format with a minimum value of 0, subtracting one value from another can lead to overflow. To address this, we adjusted the data type to **int64**.
- Managing Matplotlib windows proved challenging. Upon closing the windows, the functionality to display images using `plt.show()` appeared to be disabled, allowing only for image saving.

5.3 Conclusion

During the experiment, we generated color histograms, gray histograms, and gradient histograms for three images, extracting and analyzing their features. Additionally, we innovatively devised an edge detection method based on gradient intensity.